ProMISE - a Process Model for Information System Evolution

G. Scherrer* , A. Oberweis, W. Stucky
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe (TH)
D-76128 Karlsruhe
Germany

E-Mail: {scherrer|oberweis|stucky}@aifb.uni-karlsruhe.de

Abstract
ProMISE is a process model for evolutionary development of information systems. It is part of INCOME/STAR, an environment supporting the development and maintenance of large, distributed information systems. One of INCOME/STAR's main objectives is providing computer-support for process model usage.
Based on the criticism of existing process models, this paper summarizes the main requirements to process models. It outlines the structure and basic characteristics of ProMISE as an example for a model trying to meet these requirements.

Keywords
evolutionary software development, information system, process model, software development environment, software life cycle

1 Introduction

The development of efficient database supported information systems usually is a quite complex job as those systems are not only supposed to be specially tuned for a certain application domain but also to support a wide range of functionality within this domain. On the one hand they must be capable, e.g., to handle production control data, on the other hand they must manage business and administration data of an enterprise. In addition, systems may be integrated in networks or embedded in a heterogeneous software or hardware configuration which is subject to change.
Recently, a lot of research work has been done to achieve a deeper understanding of the *software process* which can be defined as a set of activities, methods and practices that guide people in the production of software [16]. A *process model* describes the development process in terms of *activities* and *results (documents*) and determines its overall structure (temporal order of activities, cross references between documents).
This paper starts with a critical reflection of some techniques and principles for software development (section 2), followed by a brief discussion of some new trends (section 3) which reflect key requirements for process models. Section 4 describes a conception for a process model that tries to meet these requirements: ProMISE - a process model for evolutionary development of information systems which is part of the INCOME/STAR development environment. After a brief survey of the INCOME/STAR project, the main structure and basic characteristics of ProMISE are presented. Section 5 gives an overview of the methodological support provided for the different development stages, while section 6 mentions possible future extensions.

2 Traditional Process Models

Traditionally, process models reflect the *software life cycle* with its different stages. Well-known representatives of these *stagewise* process models are Boehm's *waterfall model* [5] and its variants: they break software projects into a sequence of successive stages (requirements analysis, specification, design, code, test, operation and maintenance), in which the results of one stage serve as input documents for the following stage. Each stage contains verification and validation processes and is connected to the preceding stage by a feedback loop.

2.1 Problems

Criticism of stagewise process models mainly addresses the following aspects:

a) Lack of flexibility of sequential processes:
It has been realized that software development is a highly *dynamic* process, where different stages influence each other and cannot be rigorously separated. Although the waterfall model's feedback loop construct allows a certain deviation from a strictly sequential procedure, the flexibility gained from that is not high enough if

- the original system requirements change during development. Changes in the environment, implementation paradigms, quality assurance standards, staff and resource or a re-definition of user requirements are a common observation during realization-time of long-term projects [27].
- results of different stages affect each other (e.g. a specification has to be restricted due to implementational limitations; on the other hand, a chosen implementation may suggest an expansion of the original specification [29].)

b) Neglect of maintenance:
In spite of the well-known fact that the maintenance phase normally represents the longest and most expensive period in a software life cycle, stagewise process models regard it as a kind of appendix of the preceding development stages.

* The work of this author is partially supported by the Deutsche Forschungsgemeinschaft DFG under grant Stu 98/9 in the program "Verteilte DV-Systeme in der Betriebswirtschaft“.

Software projects are considered "completed" after the product release, and maintenance is viewed as bug-fixing. This viewpoint simply ignores the fact that new requirements may turn out during operation and that maintenance may include a rework of all prior stages.

c) Lack of attention for human factors:
Many problems in the software development field arise from communication problems between developers and future users of a system. This is a point where stagewise models have some of their most important drawbacks [19]:

- They postulate that users are sure about all system requirements right from the start. This case may be ideal but it certainly seldom holds true. Normally, the user will get a clear idea about what a system should do for him or her (and what it actually does) by the time s/he is using the system (i.e. after the release). Unfortunately, stagewise models don't offer facilities which provide users with a deeper knowledge at earlier stages.
- There is a knowledge gap between users and system designers: developers often do not know enough about the application domain to understand user requirements properly, while users may find it difficult to understand design documents.

In addition to communication problems, ergonomic reasons stand against a too restricted, strictly sequential development principle: it is still an open problem how creative subtasks inherent in every development process can be modelled (if they can be modelled at all) [27]. A process model should not restrict human creativeness too much. It should at least give the opportunity to leave things open or try different alternatives in the early stages as most humans solve complex problems using a "trial and error" policy.

2.2 Possible Solutions

The criticism mentioned above led to some new software development paradigms proposed in the eighties [1]. Concepts like *evolutionary system development, prototyping, operational specification, transformational implementation* and *software reuse* influenced software engineering. A brief survey of these concepts will be given in the following:

- **evolutionary system development:**

Software systems are subject to an intrinsic change process, i.e. they get repeatedly enhanced and extended during the operation and maintenance phase. This phenomenon is often referred as *software evolution* [17]. Evolutionary development models try to govern this change process by dividing huge software projects into a series of smaller sub-projects, each realizing a subset of the entire system's objectives. As these sub-systems are executable versions of the target system, they can be extended in a stepwise manner or enhanced considering user experiences with earlier versions.

- **prototyping:**

A *prototype* can be defined as an early version of a system which contains all basic properties of the future target system. The main goals of prototyping are validating presumed system requirements at an early stage and comparing alternate user interface layouts.
Prototyping is sometimes combined with evolutionary development principles. (For a recent example for a prototyping-based evolutionary process model cf. [4]).

- **operational specification:**

Operational specifications are interpretatively executable and may therefore be regarded as a prototype of the target system. An effective development strategy may convert this prototype into an implementation of the target system using transformational implementation methods (see next paragraph).

- **transformational implementation:**

The idea of *transformational implementation* is to generate programs automatically from a formal specification using a sequence of computer-supported transformation rules. Transformation may include generalization or specialization of the current representation. Ideally - i.e. provided that the system is constructed by an uninterrupted chain of proven transformation rules - corrections, expansions and changes will only affect the specification. Maintenance efforts can thus be reduced to keeping the specification up-to-date. (A recent approach combing the process-product-requirements model and the transformational approach can be found in [12].)

- **software reuse:**

As a last new paradigm, *software reuse* has become part of the development process [26]. Existing specifications, designs or programs contain a lot of knowledge and experience and can be a valuable basis for new systems, especially if the documents are stored in a well-organized repository.

3 Recent Trends

Regardless of the legitimacy of the above criticism, the waterfall model or similar life cycle oriented approaches have some substantial advantages: a well-structured, stagewise approach to software development can be a basis for a clearly-defined, business-like methodology, which was one of the main intentions for trying to establish software engineering as an engineering discipline.
Yet, there is still an unresolved contradiction between two key requirements to process models: on the one hand, they should result in a structured, well-planned development process; on the other hand, flexibility is required. To bridge this gap, life cycle models should be combined with some of the approaches outlined in the preceding section: working with prototypes, e.g., will definitely not make a requirements collection and analysis or other early stages obsolete. In contrast, prototyping can be extremely helpful as an integrated part of the early development stages.
An 'ideal' process model will therefore integrate many different approaches. Efforts are made to detect relationships between different areas in the field of information system engineering [18], which still lead to a deeper understanding

of software processes and process models (for a survey of experiences with process models cf. [25]). Some trends became apparent recently:

- Applications tend to be *distributed.* This means that process models must offer description facilities for a system's topology, i.e. new (or adaptive) result types are needed. Another consequence of distribution is that it suggests the use of *cooperative development techniques*, which may influence the process model's activitiy types [11, 9].
- There is a high level of agreement that information should be treated like other basic resources (e.g. monetary budget, raw materials, personnel, machines) and that the need for it should be planned independently from any particular project in a *strategy stage.* The result of this stage should be an abstract, application-independent data model which can serve as a basis for an organization-wide information system architecture [3].
- In addition to activities being directly coupled with system development, there are other activities involved in the future success of a software product: product documentation, quality assurance, user training, project and product management are highly important *side activities* [8], which should be integrated into a process model as sub-models.
- National and international *standards* for process models have been established in recent years (cf. *IEEE Standard for Developing Software Life Cycle Processes* [13], *EUROMETHOD* for EC countries [7]). Sometimes software customers – especially in public administration – require a certain public standard process model for their software development projects in order to be able to compare different offers.

4 INCOME/STAR-ProMISE: General Idea

So far, this paper has worked out key requirements for software process models. This section describes a conception for a specific process model called ProMISE that tries to meet these requirements. As mentioned before, ProMISE is a part of INCOME/STAR, a prototype of an integrated environment supporting evolutionary development of large, distributed information systems [21]. It extends INCOME, an already existing tool for conceptual modelling and prototyping of information systems, whose main features are: integration of structural and behavioural system aspects (modelled in a semantic data model and high level Petri nets), prototyping facilities and design dictionary support. Based on these concepts, a commercially available methods and tools package was developed [14].
While INCOME is primarily suited for the development of new information systems, INCOME/STAR will support both the development of completely new systems and the integration of new components into existing hardware or software environments. Special emphasis is put on distributed, heterogeneous target systems (like modern information system networks).
Our previous research work - including practical case studies - has made obvious that methods and tools packages like INCOME/STAR should offer computer support for a process model [28]. Therefore, the INCOME/STAR methodology and general insights about process model (cf. sections 2 and 3) have been condensed to a process model called ProMISE.

4.1 Basic Structure of ProMISE

Process models may have different levels of granularity. Some exist as generic process model *types* (e.g. Boehm's *spiral model* [6]), others have been refined to complete guidelines for software system development.
Figure 1 shows ProMISE on the type level, i.e. it gives an idea of its basic structure: system development with ProMISE is done in an information oriented manner, i.e. its key objective is to build a firm information structure as a basis for application development and enhancement. Hence, information system engineering and its phases (*requirements collection and analysis, conceptual modelling, database design, database implementation* and *operation*) form the centre of the overall software engineering life cycle which falls into the stages *analysis, design, implementation* and *operation.*
The circular form indicates an evolutionary approach, i.e. one iteration of the development cycle is shown. This is the reason for not having an explicit maintenance phase: in fact, maintenance is not considered as a *phase* at all, but as a *process of evolutionary system enhancement*, resulting in a new iteration of the development cycle (excluded fixing of really small bugs which can be done during operation). Generally speaking, development and maintenance of a system are done as a sequence of sub-projects, which are identified and roughly planned in a project independent, maybe organization-wide *strategic planning phase.*

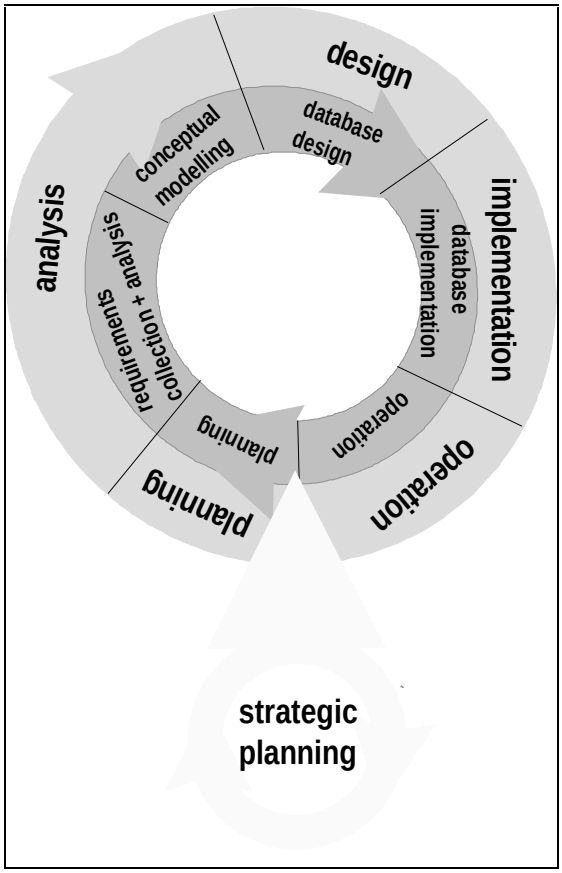

Figure 1: INCOME/STAR-ProMISE

4.2 A More Detailed View

To be applicable in practice, process models must be worked out in more detail. Therefore the general framework shown in Figure 1 has been stepwisely refined to a sequence of activities which are supported by the specific methods and tools package forming the INCOME/STAR environment.
A generic graphical representation of a refined development stage ist given in Figure 2:

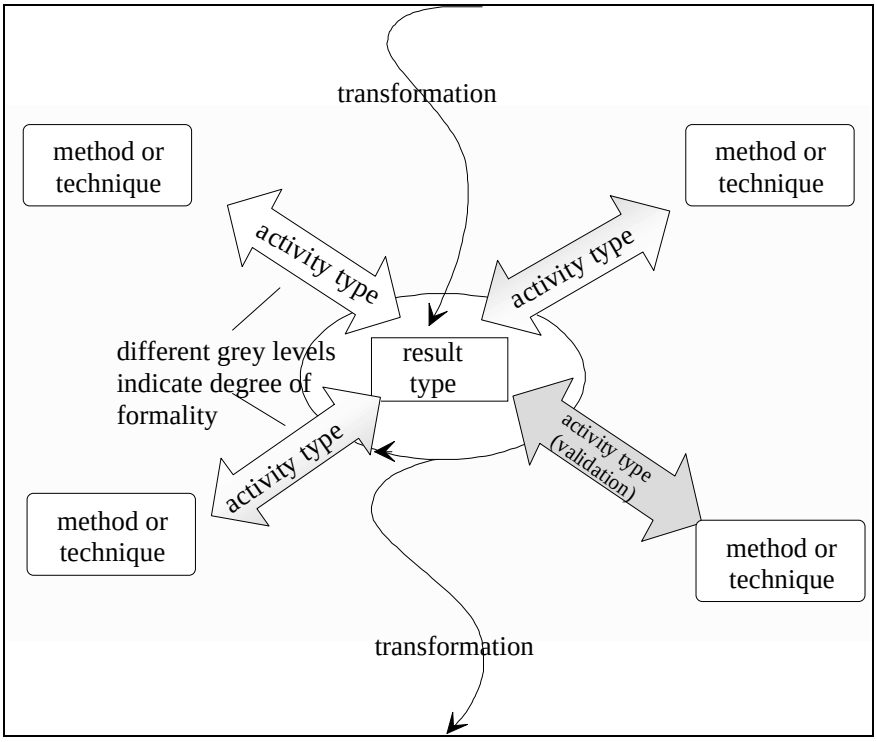

Figure 2: Generic representation of one developmnet stage in ProMISE

Each stage starts with a - more or less formal - transformation step, converting resulting documents of the preceding stage into (initial) documents of the current phase. These documents are iteratively adjusted by a circular sequence of activities (refinement, structuring, modelling, information collection and quality-checking steps etc.). Cost/benefit analysis will suggest which specific techniques should be used in a particular case, and up to which extent documents should be refined. (Note that this phase specific planning step is not mapped in Figure 2, as Figure 2 shows a fraction of the process model that maps the development process itself, i.e. it only contains activities that manipulate results directly. Planning and coordinating activities will be mapped separately in a project management component.)
Quality checks validate results of transformation and adjustment steps; if these steps lead to more than one possible result, analytical methods or simulation may be used as a decision support. Software reuse is one potential alternative - either as an integration of standard components or as a project specific adjustment of generic models.
At the end of each iteration, the existing documents will be examined in a validation or verification step. If their quality is acceptable, they may be transformed into initial documents of the successive stage. Otherwise, documents have to be modified, which means a new iteration of information collection, modification and quality checking steps.
Sometimes a situation may require even a go-back to an earlier stage, e.g. if requirements are added or changed. Generally spoken, the occurrence of irregular or unexpected events *(exceptions)* may result in a deviation from the 'regular' development process (*exception handling mechanism)*, i.e. the model is dynamically modified whenever the development process requires such dynamic changes. Other examples for exceptions are situations where deadlines cannot be met, due to, e.g., fluctuation or illness of development staff, unexpected technical failures or problems with subcontractors.
Whenever it makes sense, users will be involved into development activities. The kind of involvement must be - of course - suitable for the respective stage: In the *requirements collection and analysis stage* (Figure 5), e.g., interviews are used (to answer open questions concerning the requirements scheme) while prototyping is a technique used in the *conceptual modelling stage* (to validate the conceptual scheme, Figure 6).

5 Underlying Methodology

ProMISE may serve as a general framework for software development independently from a specific methodology (collection of methods) or CASE environment. However, a certain methodology formed the basis when ProMISE was worked out to a complete guideline for software development, i.e. existing successful methods have been completed by new concepts and brought into a reasonable order. The underlying methodologies are INCOME/Method [21], CASE*Method [3] (due to the coupling of the commercial version of INCOME to the ORACLE*CASE environment), and the new concepts which are currently worked out for INCOME/STAR. The resulting methodology will now be described in more detail. (For a summary of methods and tool support cf. table 1 in the appendix.)

- **strategic planning**

This stage aims at the collection of global strategies of an organization in order to extract a broadly-based, 'strategic' information model. Basic methods are interviews and project sessions with key people, where the current state of the (sub-) model is discussed. Since there are basic strategies to be worked out, management must be involved in the decision process at this stage. To have a reasonable basis for discussion, clear representations for the information model are essential. INCOME/STAR offers graphical descriptions: main business functions are represented by function hierarchies or by informal types of Petri nets, so-called *channel/agency nets*, where the net components are inscribed with natural language expressions. The overall information structure (main entity types and their relationships) can be expressed by a semantic data model. Cross-reference matrices show interconnections between entities and business functions. Potential applications - which may be associated with sub-projects - can be derived from the cross-reference matrices using clustering techniques.
All those description and representation methods are supported by graphical editors. The entire information contained in the diagrams is stored and related to each other in a central design dictionary, where additional information - e.g. about business units, problems, objectives, critical success factors, budget data - can be administered as well.

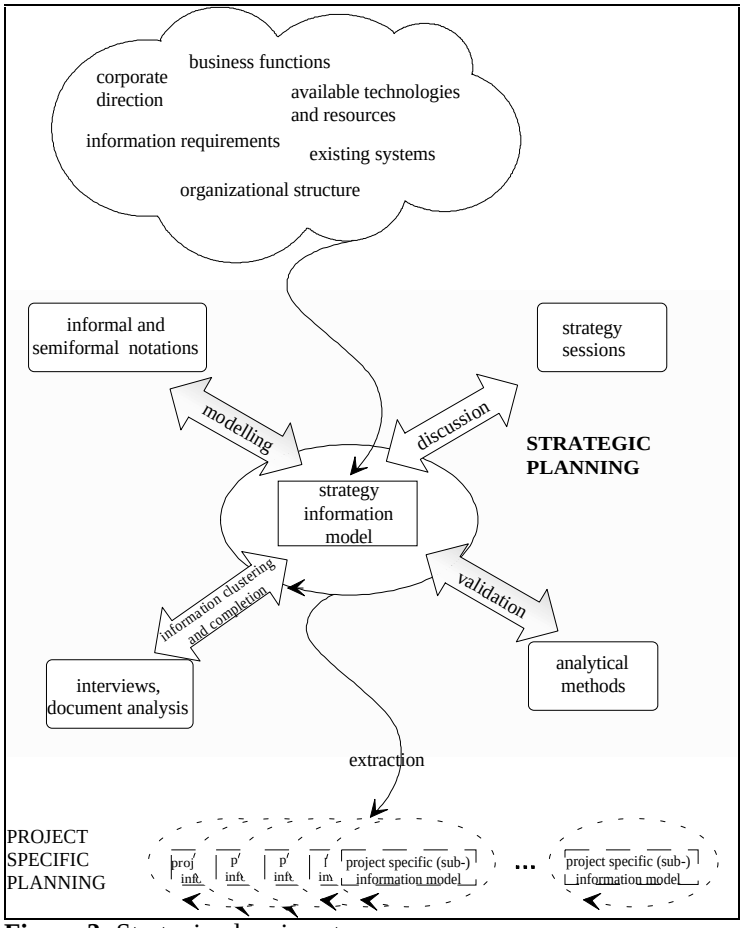

Figure 3: Strategic planning stage

- **project specific planning**

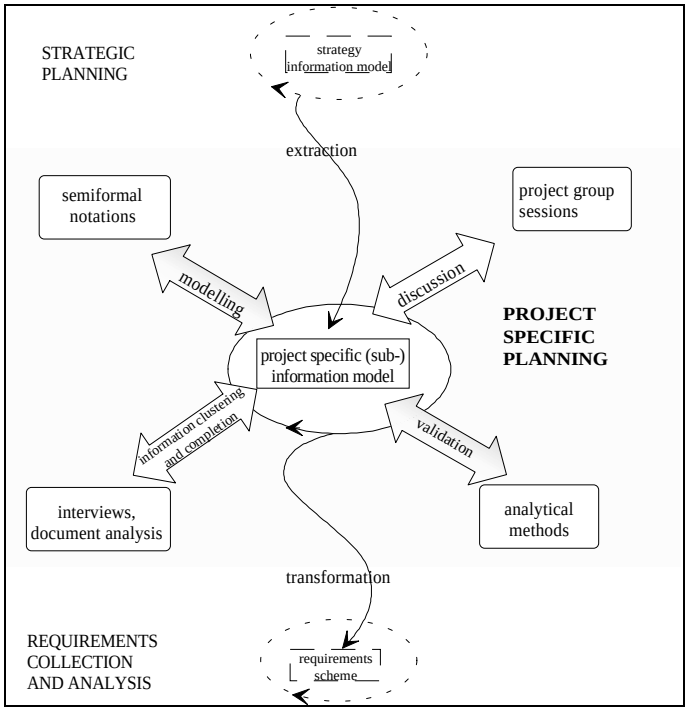

Figure 4: Project specific planning stage

the dictionary's project management facilities are planned.

Normally, this stage forms the beginning of one of the sub-projects identified in the strategic planning phase. In this case, basic functions and requirements may be derived from the information contained in the dictionary, e.g. a project specific sub-model is extracted from the strategic information model. Starting with this project specific information (sub-) model, (technical, personnel and economic) feasibility studies are carried out and different alternatives (make or buy decisions, standard software versus individual software etc.) are investigated. There may be some ad-hoc projects that had *not* been pre-planned in the strategic planning phase; in this case, an extension of the strategic information model should be taken into consideration.
Usually, the process model must be *tailored* (adjusted) to the specific needs of the specific project. One general difference, for example, is the distinction between the development of an entirely new information (sub-) system and a modification of an existing system. Time and resources are scheduled in a project management plan, that will be monitored and updated throughout the project life cycle.
Up to now, methodological support for project specific planning in INCOME/STAR is largely identical with the support offered for the strategic planning stage (graphical models and structured cross-reference matrices). Additionally, a tailoring component for the process model and an extension of

- **requirements collection and analysis**

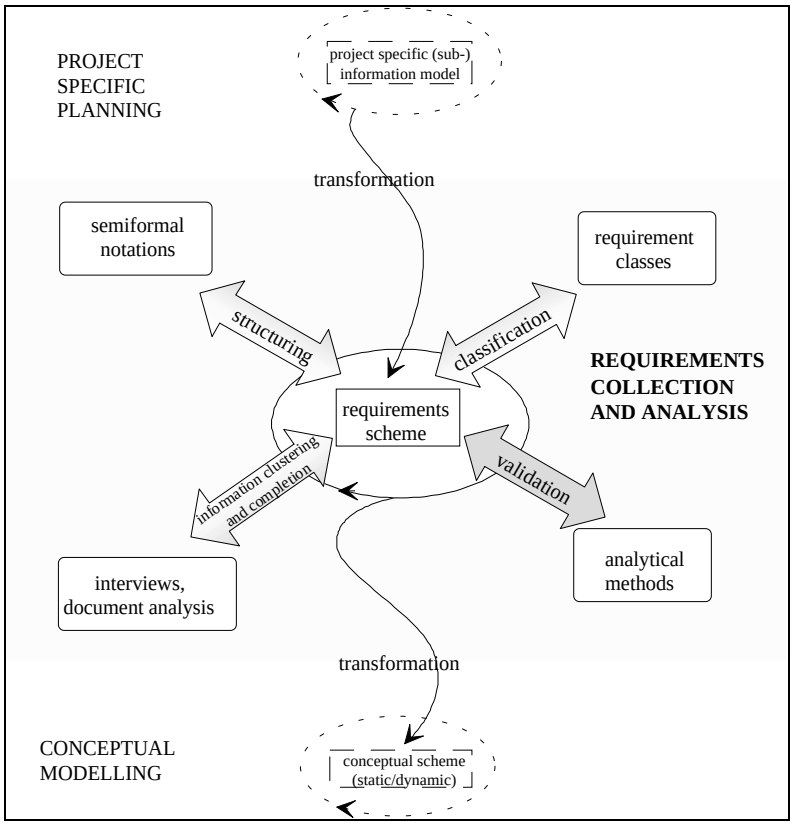

Figure 5: Requirements collection and analysis stage

Requirements collection and analysis is performed as a variant of the method proposed by [10]. In a first step, a *requirements collection plan* is worked out by extracting business units and their corresponding tasks from the project specific information model. For each task/business unit combination identified in the requirements collection plan, a requirements collection form is filled in, using interview techniques and analysis of existing forms or data files. Next, information is classified (data, operation or events requirements) and recorded in structured forms (so-called *glossaries*). Quantities of data items and frequencies of operations are estimated. To avoid redundancy and contradictions, synonyms and homonyms are eliminated from the glossary entries.

The method is supported by a special editor providing a corresponding form for each document type (collection plans, forms and glossaries) and of course all documents are part of the design dictionary. Again, cross-reference matrices provide automated quality-checking facilities validating integrity-rules like "if there exists an object x in document d1, document d2 must contain an operation y". (One possible check could make sure, e.g., that there is at least one corresponding glossary entry for each data and function element of the project specific information model.)

- **conceptual modelling**

In this stage, an initial conceptual scheme is built by converting the structured glossaries into a semantic data scheme and Petri net structures, using formal transformation rules (e.g. mapping data entries on entities and event entries on transitions). The conceptual scheme is iteratively modified and completed using bottom-up methods (generalisation, aggregation, grouping of data types, coarsening of Petri nets) as well as top-down methods (specialization, decomposition of data types, refinement of Petri nets). The INCOME/STAR toolset contains graphical editors supporting both directions for semantic data models (extended entity/relationship model [15]), high-level Petri nets and dataflow diagrams. A stepwise formalization of Petri nets leads from informal net types (channel/agency nets) to NR/T nets (nested relation/transition nets), a new variant of Petri nets allowing complex objects as tokens [23]. Additional system requirements can be modelled declaratively by so-called fact transitions restricting the set of regular states and checkpoints indicating irregular activity sequences. Exception handling mechanisms and temporal aspects (deadlines, temporal order of activities etc.) can be modelled as well.

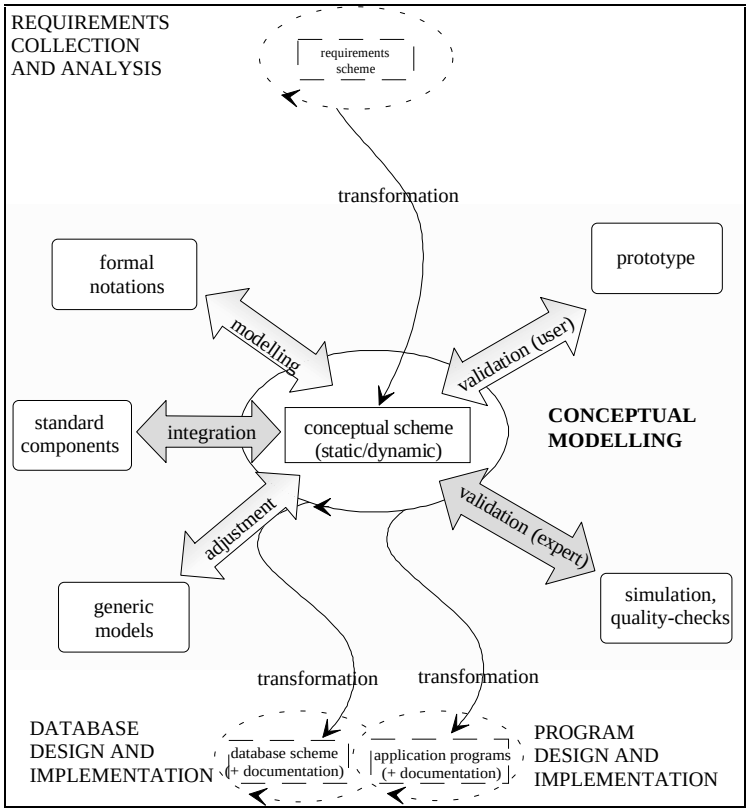

Figure 6: Conceptual modelling stage

Automated analyzers check the formal correctness of the data and function schemes and their interconnections. It is possible, e.g., to identify entities without attribute definition, isolated transitions or predicates without entity or attribute reference.

Validation of requirements is done by simulation (automated or interactive firing of transitions in Petri nets) and prototyping (Petri net simulation plus user interface similar to the target system). Results of simulation runs are stored in the dictionary and can be analyzed using a graphical query language [22].

To allow a systematic reuse of conceptual schemes, libraries containing generic models and standard components are planned. A limited amount of standard components - mostly net fragments modelling a certain temporal behaviour of activities - is available by now.

- **database design and implementation**

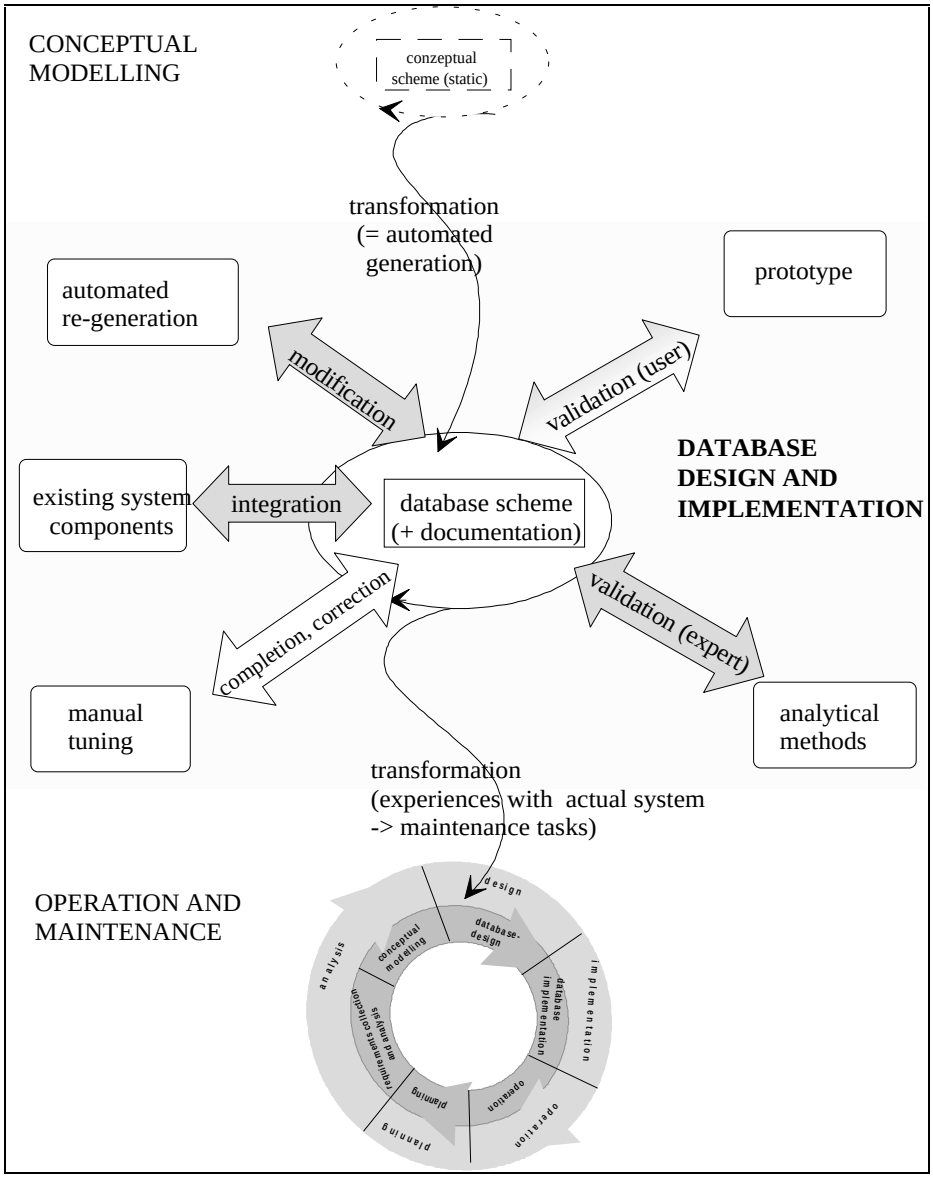

Figure 7: Database design and implementation stage

In this stage, a system architecture (which fulfills the requirements) is created. In more detail, this stage includes the following activities:

- logical design (conversion of the conceptual scheme into a relational scheme, key definition)
- (optional) design of system distribution (data and operation allocation and fragmentation, design of network architecture)
- internal design (indexing of tables, storage allocation)
- external design (user views)

A strict distinction between database design and database implementation is not possible as the availibility of 4GL generators allows a high degree of automation: Generators create normalized relations, forms, menues and reports from the semantic data scheme. The dictionary contains several default designs for this purpose, which may be modified and - if desired - kept for other projects. Possible inconsistencies (e.g. missing key definitions) are identified and eliminated.

On the function side, normalization means converting NR/T nets into Pr/T nets (predicate/ transition nets - in contrast to NR/T nets, places in Pr/T nets represent relation schemes in first normal form, not nested relations). Quantitative system aspects - e.g. performance, possible distribution concepts etc. - are investigated using either analytical methods or Petri net simulation. With respect to the design of system distribution, the use of Petri nets for behaviour modelling turns out to be a particularly useful concept, since parallel operations can be derived directly from the net structure.

A so-called *module network* showing dependencies between modules is generated from the behaviour scheme (resp. from the part of the scheme which is to be automated). Cross-reference matrices relate modules and data, while module functionality is specified in pseudo-code notation.

- **program design and implementation**

Source code generators create C-code with embedded SQL-statements from Petri nets, data schemes and cross-reference matrices [14]. If necessary, the code may be manually tuned. Existing modules are tested by coupling them to the Petri net simulator, which simulates the behaviour of connected modules.
Other generators produce reports forming the system's documentation or summarize module descriptions collected in the dictionary to a first version of the user documentation, which can be manually tuned.

- **operation and maintenance**

We define maintenance as the set of modifications performed after the release. It has been stated previously that ProMISE does not take maintenance as a stage in the development cycle but as a new iteration of the cycle. (Therefore, there is no graphical representation of this stage.)
In the planning stage, registered maintenance requests are investigated and - if accepted - grouped into maintenance projects. The process model is tailored for the specific maintenance project; some activities may become obsolete, new ones may be added. Yet, the principle process structure and the methodology will be mainly the same: during new development, existing documents have to be modified or extended as well - for example if prototyping methods or quality checks identify errors or insufficient realization of require-

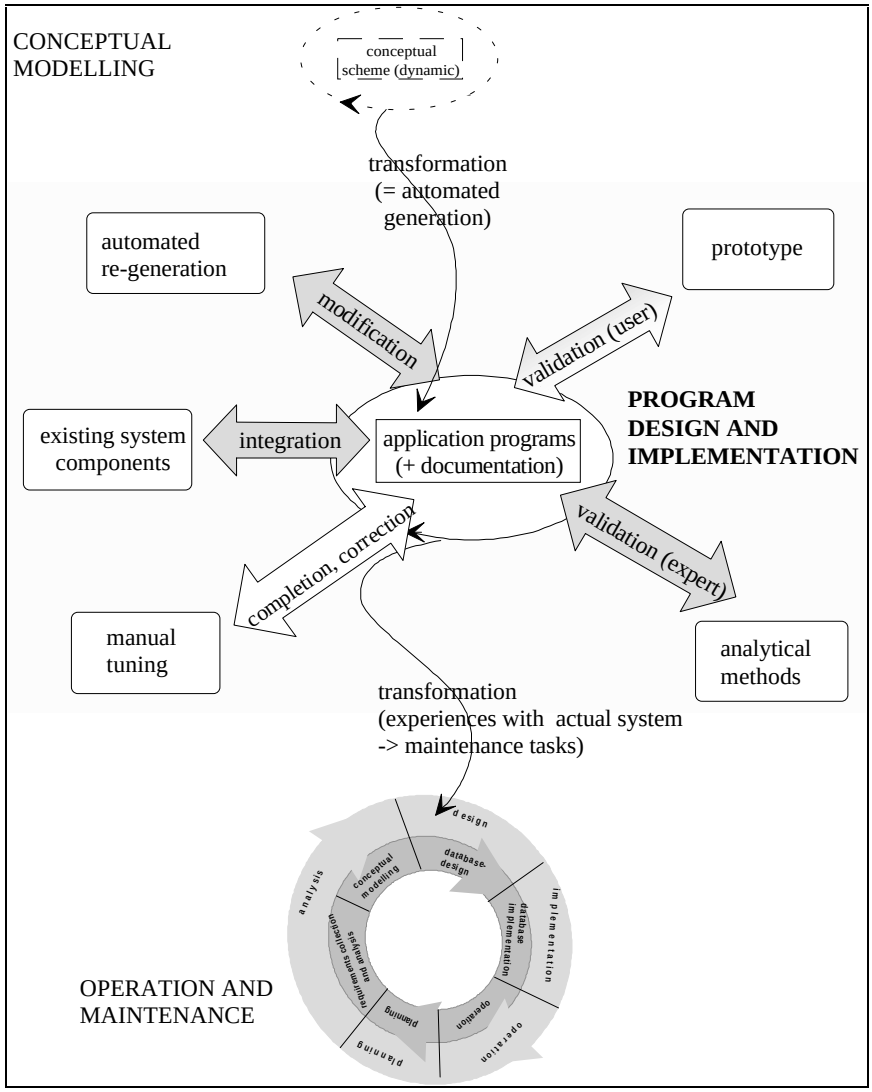

Figure 8: Program design and implementation stage

ments. The main difference to maintenance modifications is that one normally does not have to consider documents of successive stages simply because they do not exist. In other words, maintenance can be thought of as development under consideration of certain restrictions given by an existing target system. Hence, there are two important requirements to an environment supporting maintenance:
a) During analysis it must be thoroughly investigated which parts of the system will be affected by a change (impact analysis). To perform this, the dictionary with its various stage-overlapping analyzers gains particular importance. Additionally, a hypertext interface is planned [20] to provide an even better visualization of stage-overlapping interrelations by connectiong, e.g. source code with corresponding requirements or design documents or parts of the documentation.
Simulation as a tool for validation and comparison of modification proposals is another concept of INCOME/STAR which supports impact analysis.
b) The existing system restricts developers' opportunities. It would be useful if such restrictions could be expressed in a simple, precise fashion. With the concept of checkpoint transitions (cf. section 'conceptual modelling'), a powerful means for the expression of such constraints is available.

6 Summary and Outlook

This paper outlined the main structure and basic characteristics of ProMISE, a process model for development and maintenance of information systems, which is part of the INCOME/STAR environment. The main features of the model are: an integration of different development principles like user participation, automated software generation, a stage-integrating, computer-supported methodology, software reuse and design-dictionary support.
At this time, documents and activity structures are further refined to a detailed formal description based on Petri nets and semantic data schemes, permitting an integration of the process model into the design dictionary. Main benefits will be an enhanced efficiency of process model usage and the opportunity to map different process model standards on each other.

Future research work will largely concentrate on the following fields: At the moment, *project management* support is more or less limited to the administration of some budget dates and critical success factors in the dictionary. Future efforts in this area will concern the development of a detailed sub-model for computer-supported resource, capacity, time and cost planning.
Many applications require integration of database and knowledge base components. Therefore, it could be useful to combine process models from the area of *knowledge engineering* and of software engineering. We are currently investigating possibilities to integrate concepts from MIKE (**M**odel-based and **I**ncremental **K**nowledge **E**ngineering [2] which is also developed at our institute) and INCOME/STAR-ProMISE.
A further research area is cooperative system development. Currently, INCOME/STAR provides simple locking-mechanisms, future efforts will put stronger attention on c*omputer supported cooperative work* techniques like providing an intelligent mailbox concept to support communication between developers [24].

Appendix

Stage	Methods	Tool support
Strategic information planning	extended ER-model channel/agency net hierarchies function hierarchies cross-reference matrices structured glossaries	graphical editors automated analyzers
Project specific planning	extended ER-model channel/agency nets hierarchies function hierarchies cross-reference matrices structured glossaries	graphical editors automated analyzers tailoring-component
Requirements collection and analysis	extended ER-model dataflow diagrams channel/agency nets hierarchies function hierarchies cross-reference matrices structured glossaries	glossary editor graphical editors automated analyzers
Conceptual modelling	extended ER-model NR/T nets: • hierarchies • declaratively modelled system requirements • time aspects standard components generic models simulation of design alternatives	graphical editors NR/T net generators automated analyzers libraries for standard components and generic models simulators: • quantitative investigations • graphically animated
Logical design (→ rel. data base)	algorithms SQL-Pr/T-Nets simulation of design alternatives	graphical editors SQL-Pr/T net generators automated analyzers
Distribution design	algorithms for data allocation and fragmentation: • generation of design alternatives • valuation (analytic/simulation-based)	automated analyzers simulators (distributed simulation model)
Ext. / int. data base design	algorithms	automated analyzers
Program design & implementation	automated generation	data base generators source code generators
Operation and maintenance	algorithms simulation of modification proposals hypertext techniques	graphical editors simulators hypertext interface stage-overlapping automated analyzers: • cross-checks • correlation analysis • consistency checks

Table 1: INCOME/STAR-ProMISE: methods and tools

Literature

[1] W.W. Agresti (Ed.): *New Paradigms for Software Development,* IEEE Computer Society Press, Washington 1986

[2] J. Angele, D. Fensel, D. Landes, S. Neubert, R. Studer: Model-based and incremental knowledge engineering: the MIKE approach, in: J. Cuena (Ed.): Proc. IFIP TC12 Workshop on Artificial Intelligence from the Information Processing Perspective, Madrid 1992, Elsevier, Amsterdam 1993

[3] R. Barker: *CASE*Method: Tasks and Deliverables,* Addison-Wesley, Wokingham, 1990

[4] A.T. Berztiss: Concurrent engineering of information systems, in: N. Prakash, C. Rolland, B. Pernici (Eds.): *Information System Development Process*, Proc. of the IFIP WG8.1 Working Conference on Information System Development Process, Como, Italy, September 1993

[5] B.W. Boehm: Software Engineering, *IEEE Transactions on Computers*, 25, p. 1226-1241, 1976,

[6] B.W. Boehm: A spiral model of software development and enhancement, *IEEE Computer*, 21(5), p. 61-72, May 1988

[7] Commission of the European Communities: EUROMETHOD Information Pack, Brussels 1991

[8] G. Chroust: *Modelle der Software-Entwicklung*, R. Oldenburg Verlag, Munich, Vienna 1992 (in German)

[9] B. Curtis, D. Walz, J. Elam: Studying the process of software design teams, in: D.E. Perry (Ed): *Experiences with Software Process Models*, Proc. 5th International Software Process Workshop, Kennebunkport, Maine 1989, IEEE Computer Society Press, Los Alamitos 1990

[10] V. DeAntonellis, B. Demo: Requirements collection and analysis, in: S. Ceri (Ed.): *Methodology and Tools for Data Base Design*, p. 9-24, North-Holland Publ. Comp., Amsterdam, New York 1983

[11] A. Finkelstein: A structural framework for the formal representation of cooperation, in: D.E. Perry (Ed): *Experiences with Software Process Models*, Proc. 5th International Software Process Workshop, Kennebunkport, Maine 1989, IEEE Computer Society Press, Los Alamitos 1990

[12] J.-L. Hainaut, M. Cardelli, B. Decuyper, O. Marchand: Database CASE tool architecture: principles for flexible design strategies, in: P. Loucopoulos (Ed.): *Advanced Information Systems Engineering*, Proc. 4th International Conference CaiSE '92, Manchester 1992, Springer-Verlag, Berlin, Heidelberg, New York 1992

[13] IEEE Standard for Developing Software Life Cycle Processes, Institute of Electrical and Electronics Engineers, New York, January 1992

[14] INCOME User Manuals: *INCOME/Designer*, *INCOME/Dictionary*, *INCOME/Generator*, *INCOME/Simulator*. PROMATIS Informatik, Karlsbad 1993

[15] P. Jaeschke, A. Oberweis, W. Stucky: Extending ER model clustering by relationship clustering, in: R. Elmasri, V. Kouramajian (Eds.): Proc. 12th International Conference on the Entity Relationship Approach, p. 447-459, Arlington/Texas, December 1993.

[16] D.H. Kitson, S.M. Masters: An analysis of SEI software process assessment results: 1987-1991, Proc. 15th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, California, May 1993, p.68-77

[17] M.M. Lehman: Programs, life cycles and the laws of software evolution, *Proc. of the IEEE*, 68(9), p. 1060-1076, September 1980

[18] P. Loucopoulos (Ed.): *Advanced Information Systems Engineering*, Proc. 4th International Conference CaiSE '92, Manchester 1992, Springer-Verlag, Berlin, Heidelberg, New York 1992

[19] M.M. Mantei, T.J. Teorey: Cost/benefit analysis for incorporating human factors in the software life cycle, *Communications of the ACM*, 31(4), p. 426-439, April 1988

[20] S. Neubert, A. Oberweis: Einsatzmöglichkeiten von Hypertext beim Software Engineering und Knowledge Engineering, in: Proc. Hypertext und Hypermedia 92, Munich, p.162-174, September 1992 (in German)

[21] T. Németh, A. Oberweis, F. Schönthaler, W. Stucky: INCOME: Arbeitsplatz für den Programmentwurf interaktiver betrieblicher Informationssysteme, Forschungsbericht 251, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, August 1992 (in German)

[22] A. Oberweis, V. Sänger: Graphical query facility for large Petri net simulation runs, in: F. Maceri (Ed.): Proc. of the 1992 EUROSIM Conference, Capri/Italy, p. 515-520, September/October 1992

[23] A. Oberweis, P. Sander, W. Stucky: Petri net based modelling of procedures in complex object database applications, in: J.E. Urban (Ed.): Proc. 17th Annual International Computer Software and Applications Conference COMPSAC 93, Phoenix/Arizona, p. 138-144, November 1993

[24] A. Oberweis, W. Stucky, T. Wendel: Rechnergestützte Kommunikation in Software-Entwicklungsprojekten - Workgroup Computing für kooperative Sytementwicklung. Proc. Online '94, 17. Europäische Congressmesse für Technische Kommunikation, February 1994 (in German, to appear)

[25] D.E. Perry (Ed): *Experiences with Software Process Models*, Proc. 5th International Software Process Workshop, Kennebunkport, Maine 1989, IEEE Computer Society Press, Los Alamitos 1989

[26] N. Prakash, C. Rolland, B. Pernici (Eds.): *Information System Development Process*, Proc. of the IFIP WG8.1 Working Conference on Information System Development Process, Como, Italy, September 1993

[27] G. Starke: Urgent research issues in software process engineering, *ACM SIGSOFT Software Engineering Notes*, 18(4), p. 13-15, October 1993

[28] W. Stucky, A. Oberweis, G. Scherrer: INCOME/STAR: Process model support for the development of information systems, in: J. Niedereichholz, W. Schuhmann (Eds.): *Wirtschaftsinformatik - Beiträge zur modernen Unternehmensführung*, p. 145-165, Campus-Verlag, Frankfurt/New York 1993

[29] W. Swartout, R. Balzer: On the inevitable intertwining of specification and implementation, *Communications of the ACM*, 25(7), p.438-440, July 1982